monokle

Edit, Debug, and Secure K8s Manifests

Why it’s important and how to get it right

Ole Lensmar / kubeshop.io / ole@kubeshop.io

monokle

Agenda

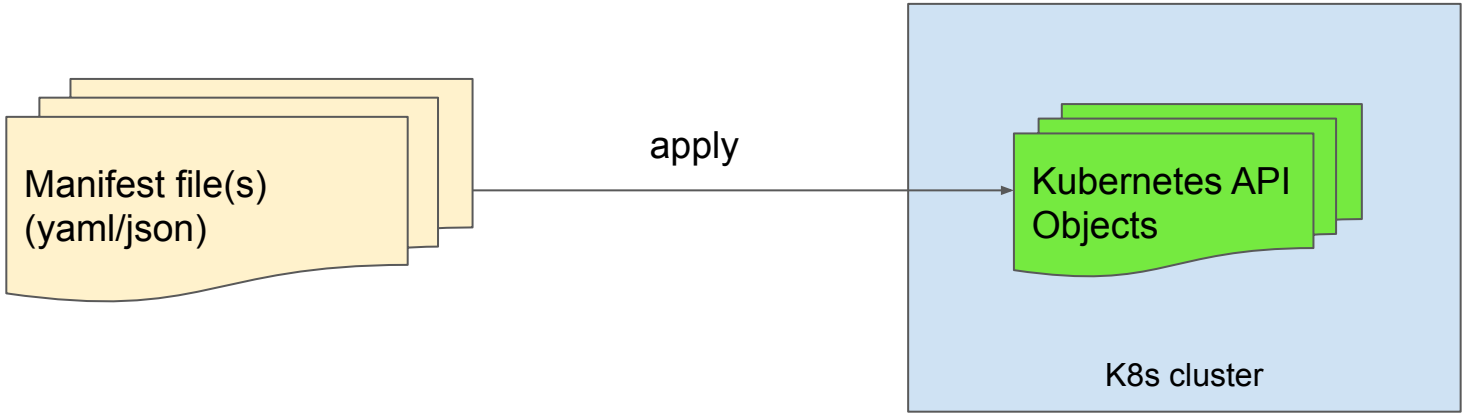
- Manifest 101 - what are they?
- The manifest lifecycle
- Creating / validating manifests
- Templating manifests
- Tooling
- Best Practices

monokle

What are Kubernetes manifests?

Specification of a Kubernetes API object in JSON or YAML format.

A manifest **specifies the desired state** of an object that Kubernetes will maintain when you apply the manifest. A configuration file can contain multiple manifests.



monokle

Basic Manifest Structure

- `apiVersion` - the API version
- `kind` - the type of K8s object
- `metadata`
  - `name` - the name of the object
  - `namespace` - the target namespace (optional)
  - `labels` and `annotations` (optional)
- kind-specific content
  - specifies the desired state of the object to be created

Example Service

```yaml
---
apiVersion: v1
kind: Service
metadata:
  name: petstore
  namespace: default
spec:
  ports:
    - name: http
      port: 80
      targetPort: 8080
  selector:
    app: petstore
```

monokle

Manifest versions and schemas

- The `apiVersion` and `kind` of the manifest specifies which *schema* to use
- The *schema* defines the “content” of the manifest - i.e. the state of the described resource - properties, arrays/maps, types, enumerations, etc.
  - Uses JSON Schema as used by OpenAPI 3.0 - with certain limitations
  - API documentation available at kubernetes.io

- Not to be confused with the version of Kubernetes itself!
  - Specific versions of Kubernetes support specific apiVersions
  - Example: Kubernetes 1.24 supports both apiVersion “v1” and “v1beta1” of the CronJob kind
    - (“v1beta1” will be removed in Kubernetes 1.25)

monokle

Manifest relationships

- K8s objects often reference other objects
  - Name references
  - Label-based selectors
  - Object references

```yaml
24        env:
25        - name: ARGOCD_SERVER_INSECURE
26          valueFrom:
27            configMapKeyRef:
28              name: argocd-cmd-params-cm
29              key: server.insecure
30              optional: true
31        - name: ARGOCD_SERVER_BASEHREF
```

```yaml
1  ---
2  apiVersion: rbac.authorization.k8s.io/v1
3  kind: RoleBinding
4  metadata:
5    creationTimestamp: 2022-06-03T08:58:54.000Z
6    name: kubeadm:bootstrap-signer-clusterinfo
7    namespace: kube-public
8    resourceVersion: "235"
9    uid: 0892ce48-de31-4608-8ef8-6c46d2879eb4
10 roleRef:
11   apiGroup: rbac.authorization.k8s.io
12   kind: Role
13   name: kubeadm:bootstrap-signer-clusterinfo
14 subjects:
15   - apiGroup: rbac.authorization.k8s.io
16     kind: User
17     name: system:anonymous
18
```

```yaml
1  apiVersion: v1
2  kind: Service
3  metadata:
4    labels:
5      app.kubernetes.io/name: argocd-metrics
6      app.kubernetes.io/part-of: argocd
7      app.kubernetes.io/component: metrics
8    name: argocd-metrics
9  spec:
10   ports:
11   - name: metrics
12     protocol: TCP
13     port: 8082
14     targetPort: 8082
15   selector:
16     app.kubernetes.io/name: argocd-application-controller
17
```

monokle

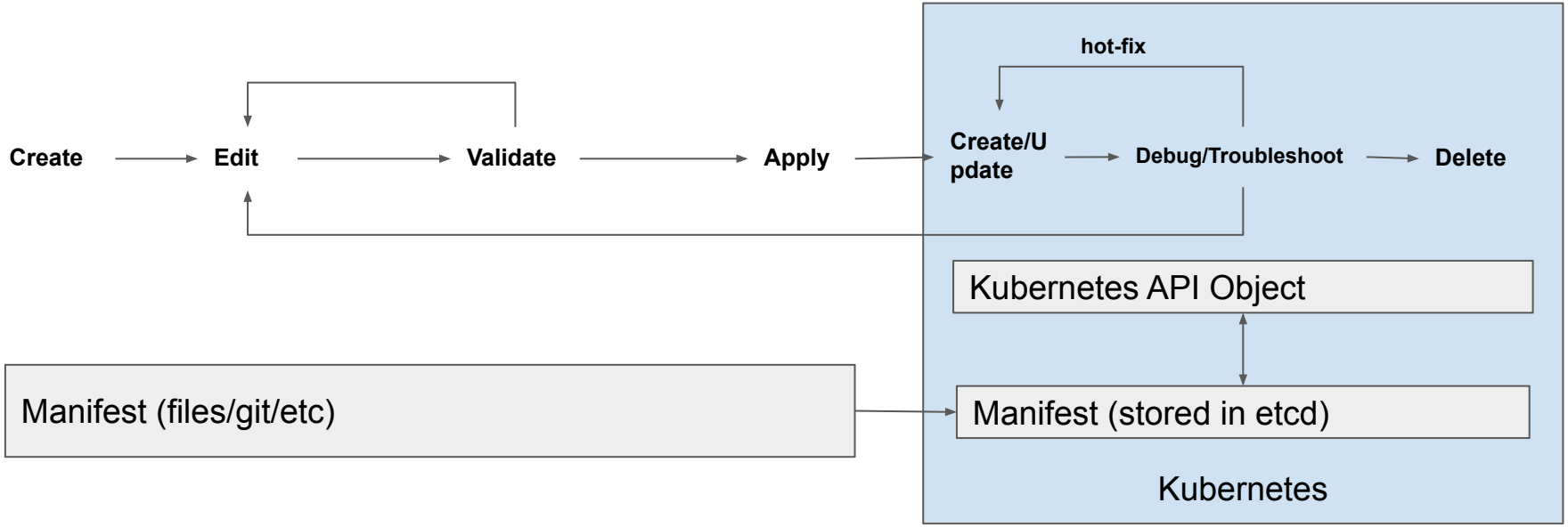
Manifest Status

- Once deployed to your cluster Kubernetes adds a `status` property to the manifest describing the current state of the object
  - continuously updated by the Kubernetes system and components
  - (not for all object types)


```yaml
status:
  conditions:
    - lastProbeTime: null
      lastTransitionTime: 2022-06-03T08:59:33.000Z
      status: "True"
      type: Initialized
    - lastProbeTime: null
      lastTransitionTime: 2022-08-03T13:08:47.000Z
      status: "True"
      type: Ready
    - lastProbeTime: null
      lastTransitionTime: 2022-08-03T13:08:47.000Z
      status: "True"
      type: ContainersReady
    - lastProbeTime: null
      lastTransitionTime: 2022-06-03T08:59:33.000Z
      status: "True"
      type: PodScheduled
  containerStatuses:
    - containerID: docker://d579f4f4f98ec598610ada90ca49b265024dd7863561672504
      image: swaggerapi/petstore3:unstable
      imageID: docker-pullable://swaggerapi/petstore3@sha256:c64f92614742aeda0
      lastState:
        terminated:
          containerID: docker://a34a46c5bfa5784711c3e86ec2e406cfce48504e0b233f
          exitCode: 255
          finishedAt: 2022-08-03T13:08:27.000Z
          reason: Error
          startedAt: 2022-08-03T10:16:41.000Z
      name: service
      ready: true
```

monokle

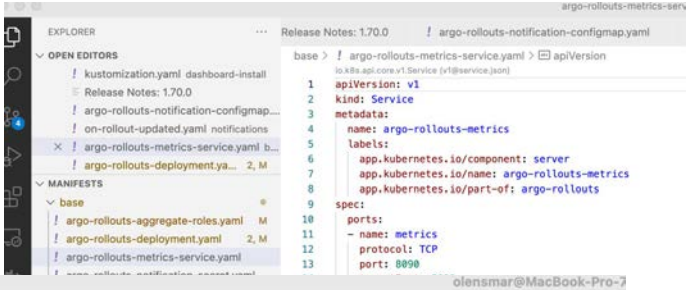
The Kubernetes Manifest lifecycle



monokle

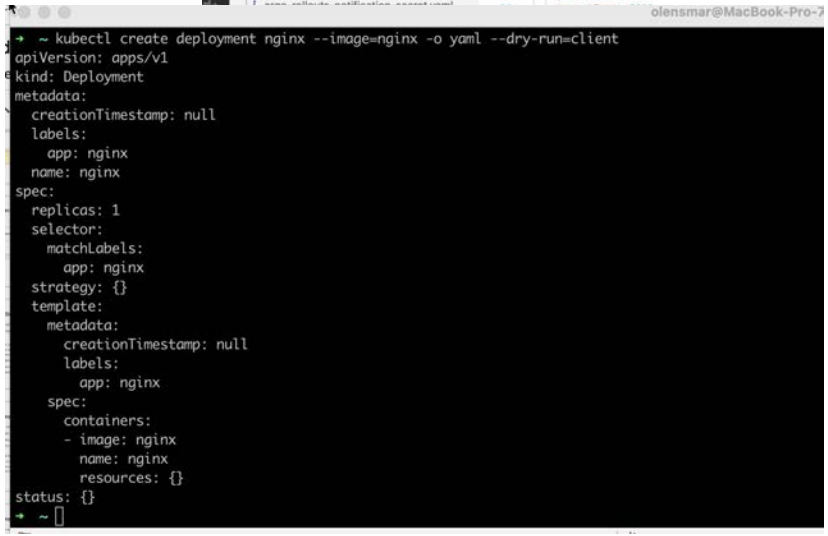
Creating / editing manifests

- Create / edit in your IDE
  - Copy/paste from another manifest (if you dare..)
  - Use plugins / code-snippets / generators



- Use kubectl

```
kubectl create deployment nginx
--image=nginx -o yaml
--dry-run=client
```



monokle

Validating manifests

1. Syntax validation
   - Valid YAML/JSON
2. Schema validation
   - Required & Valid properties / values
3. Link validation
   - Valid references to other objects
4. Policy validation:
   - Local, Performance, Security, etc.

```
216          runAsNonRoot: true
217          capabilities:
218            drop:
219              - all
220        volumes:
221        - emptyDir: {}
222          names: plugins-home
223        - emptyDir: {}
224          name: tmp
225        - name: ssh-known-hosts
226          configMap:
227            name: argocd-ssh-known-hosts-cm
228        - name: tls-certs
229          configMap:
230            name: argocd-tls-certs-cm
231        - name: argocd-repo-server-tls
232          secret:
233            secretName: argocd-repo-server-tls
234            optional: false
235            items:
236            - key: tls.crt
```

monokle

Validating manifests with OPA (Open Policy Agent)

- OPA Used by several open-source projects for Kubernetes configuration validation
  - Applied either before deploying or as part of the deployment process
  - Rules written in rego

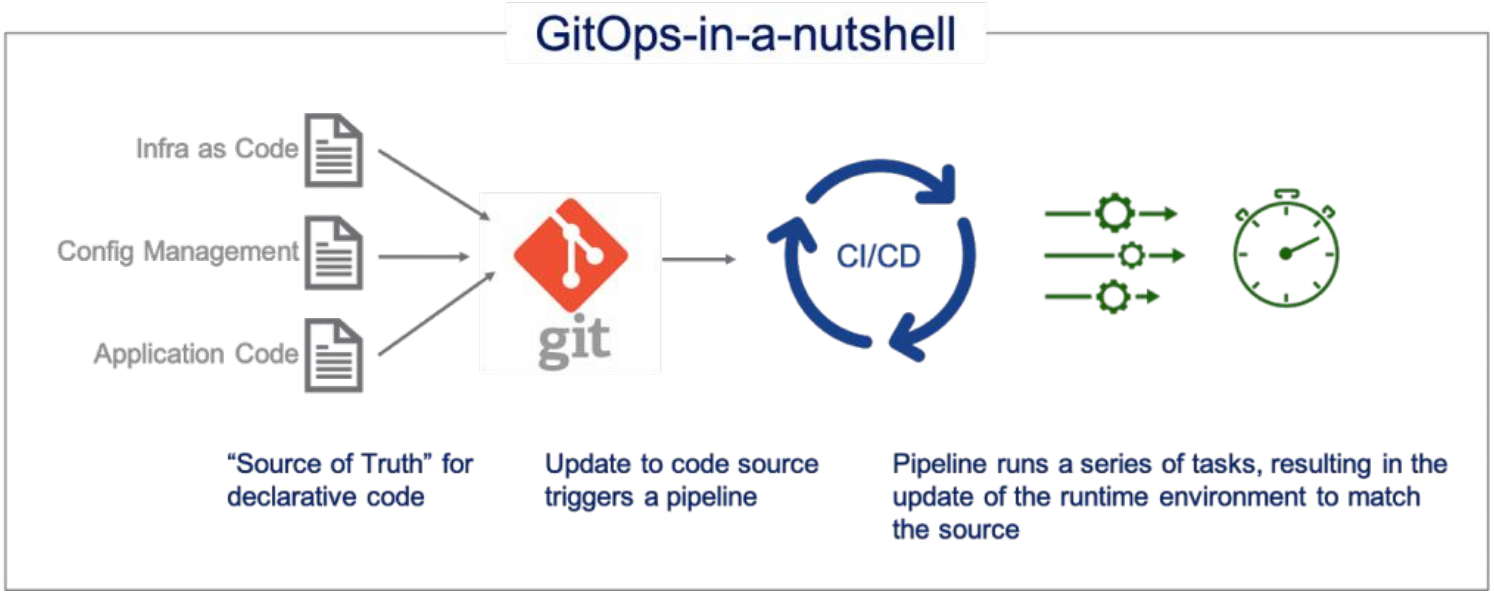
- Can be used to validate any configuration aspect of a resource
    - Names, labels
    - Network configuration
    - Resource allocation
    - Custom - “Best practices”

- Many predefined rules available (GitHub, etc)
  - VS-Code plugin available for creating your own

monokle

Finally - deploying manifests to your cluster

- Manually: kubectl / helm / kustomize / etc
- Automated: CI/CD - GitOps



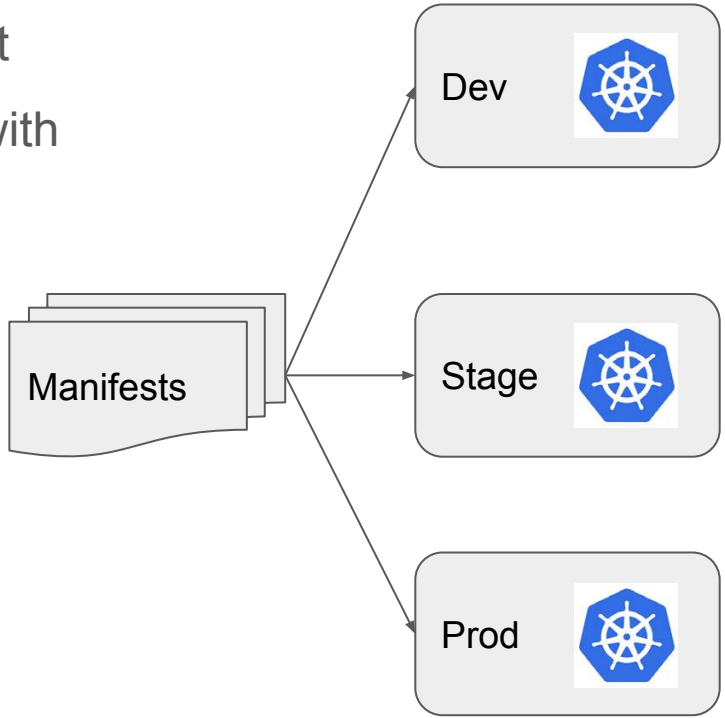
Let’s have a look!

monokle

Manifest Templating

- **Need**: have a common set of manifests that can be deployed to different environments with different parameters

- Different approaches:
  - YAML-native (Kustomize, yq)
  - Custom templating (Helm, Jsonnet, etc)
  - Generate from code (cdk8s, dekorate, etc)
  - Abstraction layers (Acorn, etc)



monokle

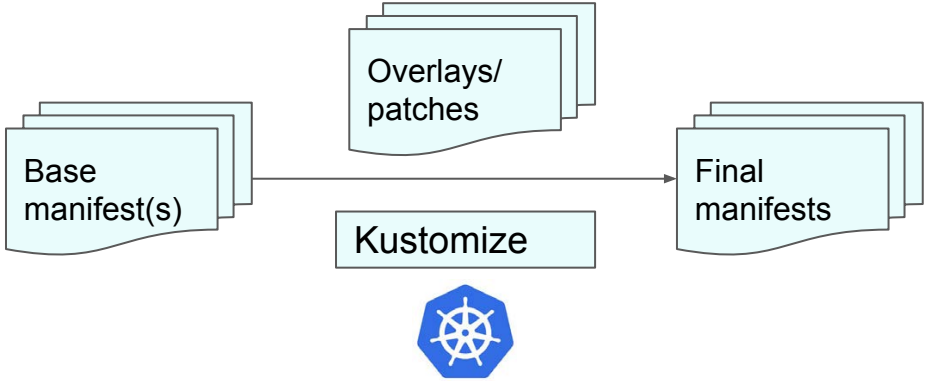
YAML-Native : Kustomize

- “Kubernetes native configuration management” - kustomize.io
- Uses plain YAML for templating/patching
- Built into kubectl

Kustomize File Structure

Environment Example

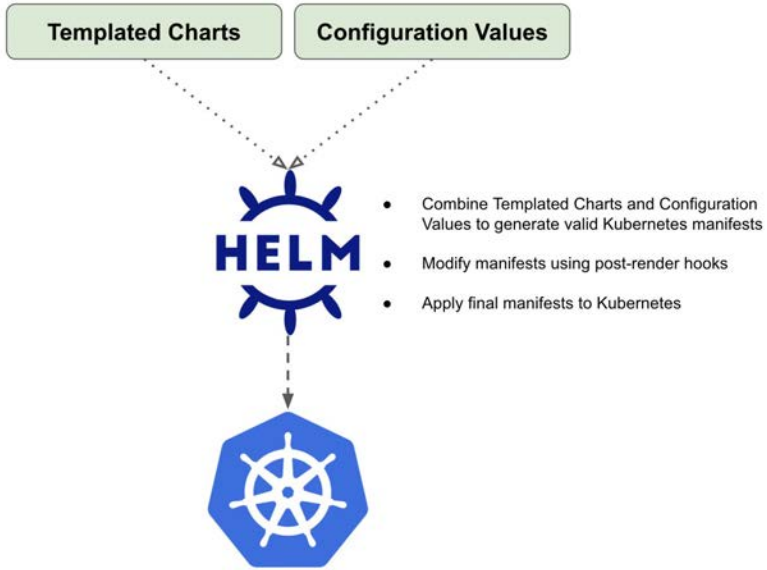
```
hello-world/
├── base
│   ├── deployment.yaml
│   └── kustomization.yaml
└── overlays
    ├── production/
    │   ├── replica-count.yaml
    │   └── kustomization.yaml
    ├── dev/
    │   ├── replica-count.yaml
    │   └── kustomization.yaml
```



monokle

Custom Templating: Helm

- Helm uses custom templates packaged into “Charts” for packaging applications
- A “Helm Chart” produces a set of Kubernetes manifests to be deployed to a cluster
- Helm Charts can be parameterized using values file(s) as configuration input
- Helm Charts are distributed/consumed via “Helm Repositories”



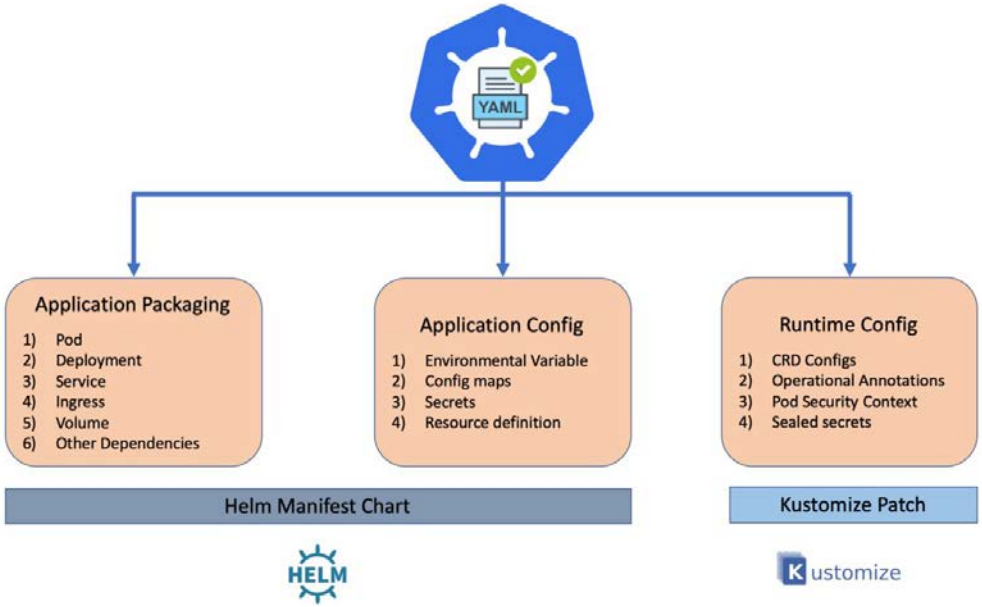
monokle

Using Helm and Kustomize together

- Helm and Kustomize complement each other nicely!

- Use Helm to package your application

- Use Kustomize to manage runtime configuration



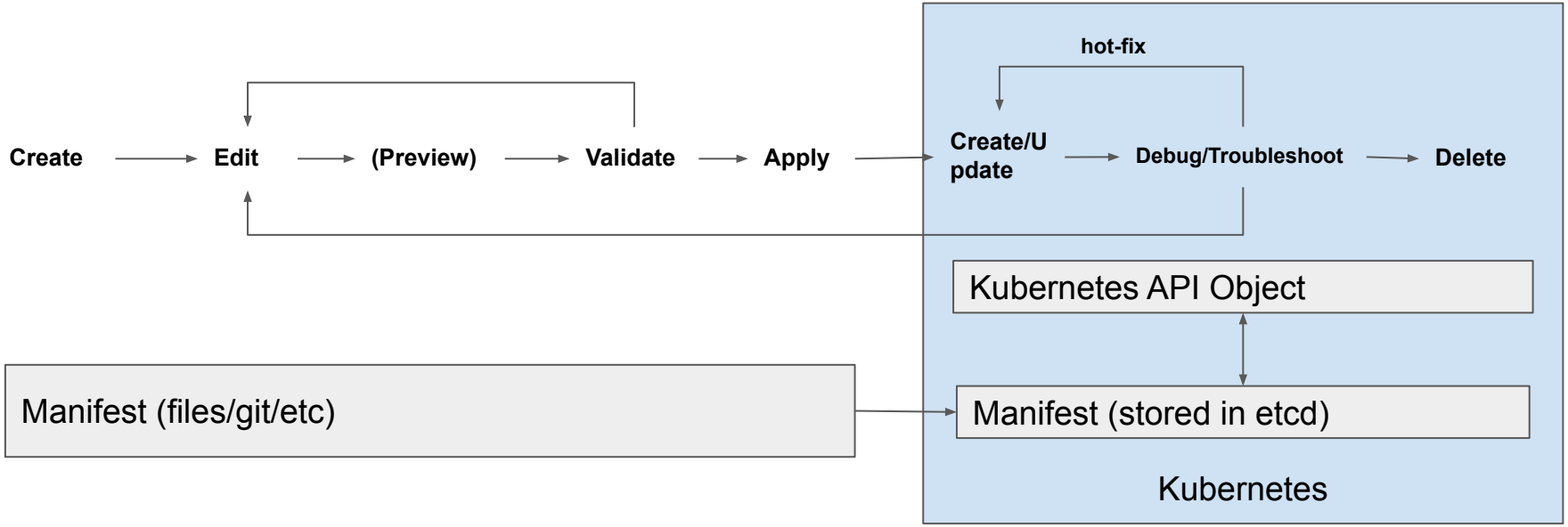
monokle

Templating and the Manifest lifecycle

- Create/Edit as before - using your favorite IDE / Code editor
- “Dry-run” templating tools to perform validations
- Required to inspect / validate generated manifests
  - Compliance with target Kubernetes version
  - Policies / Security
  - Cluster references
- Automate as part of pre-deployment checks

monokle

The Kubernetes Manifest lifecycle - revisited



Short Kustomize Demo

monokle

Tooling for the Kubernetes Manifest lifecycle

	Manifest Lifecycle					Kubernetes Object Lifecycle		
	Create	**Edit**	**Preview**	**Validate**	**Apply/Deploy**	**Create/ Update**	**Debug**	**Delete**
kubectl	X				X	X		X
IDEs + plugins (VS-Code, Intellij, etc)	(X)	X		(X)	(X)	(X)	(X)	(X)
Validation tools (Trivvy, Kubescape, etc)				X				
Templating tools (Helm, Kustomize, etc)	X		X		X	X		X
Cluster inspection tools (Lens, k9s, etc)					(X)	X	X	X
Kubernetes IDEs (Monokle, etc)	X	X	X	X	X	X	X	X

monokle

Getting it right - Best Practices

1. Understand manifests and their lifecycle
2. Use latest stable API Version for your manifests
3. Keep manifests simple (do not specify default values unnecessarily)
4. Define project/team policies for resource configurations and metadata
5. Use templating - when you need it
6. Validate manifests *before* you deploy
7. Automate manifest validation and deployment as part of CI/CD
8. Adopt GitOps - when everyone understands the implications

monokle

Q&A

Thanks for listening!

ole@kubeshop.io